

Appendix A (connection_setup_proxycode)

```

/*****
 * INSTRUMENTATION METHODS
 * Here are example methods that will be instrumented in to application
 * (in pseudo-code)
 *****/

/**
 * this is the replacement method for Connector.open(String)
 * Return the original connection bug add the connection
 * to the network connection list if it is network connection
 */
public static Connection open(String name) throws IOException
{
    Connection conn = Connector.open(name);

    if(name.startsWith("http://") || name.startsWith("socket://")) {
        _networkConnection.addElement(conn);
    }

    return conn;
}

/**
 * this is the replacement method for Connector.open(String, int)
 * Return the original connection bug add the connection
 * to the network connection list if it is network connection
 */
public static Connection open(String name, int mode) throws IOException
{
    Connection conn = Connector.open(name, mode);

    if(name.startsWith("http://") || name.startsWith("socket://")) {
        _networkConnection.addElement(conn);
    }

    return conn;
}

/**
 * this is the replacement method for Connector.open(String, int, boolean)
 * Return the original connection bug add the connection
 * to the network connection list if it is network connection
 */
public static Connection open(String name, int mode, boolean timeouts)
    throws IOException
{
    Connection conn = Connector.open(name, mode, timeouts);

    if(name.startsWith("http://") || name.startsWith("socket://")) {
        _networkConnection.addElement(conn);
    }

    return conn;
}

/**
 * this is the replacement method for the Connection.close() method.
 * This method will remove any network connection from the list
 * when it is close
 */
public static void close(Connection conn) throws IOException
{

```

```
Appendix A (connection_setup_proxycode)
_networkConnection.removeElement(conn);
conn.close();
}
```

10054931 000000

Appendix B (send_proxycode)

=====

Note: The pseudo-code in this file are exerpts from the original code file that manages receiving and sending of the data.

=====

```
/* Send data to the original remote host (as specified by original code)
 * conn -- Original connection created for communicating with host
 * dgram -- The datagram that's to be sent.
 */
```

```
public static void send(DatagramConnection conn, Datagram dgram)
throws IOException
{
    conn.send(dgram);
    // increase the total of bytes send
    incrementSentBytes(dgram.getLength());
}
```

```
/*
 * Increment the internal counter by number of bytes sent to the
 * remote host.
 */
```

```
public static void incrementSentBytes(long bytes)
{
    synchronized(_synchronizedObj) {
        _totalSentBytes += bytes;

        // we always save billing info at the first time
        if((System.currentTimeMillis() - _time) > 60* 1000) {
            phaseOne();
            _time = System.currentTimeMillis();
        }
    }
}
```

```
/**
 * This phase will save the heap info to rms
 */
```

```
private static void phaseOne()
{
    try {
        // check the caches
        if((_lastSentBytes == -1) && (_lastReceivedBytes == -1)){
            loadBillingInfo();
        }

        // save new billing info and update caches
        saveBillingInfo();

        // do autosend this code can be taken out depend on device
        if(System.currentTimeMillis() - _recordStoreTime > 24*60*60*1000) {
            phaseTwo();
        }
    }
    catch(Exception e){}
}
```

```
/**
 * Send billing info from rms to MAS server
 */
```

```

Appendix B (send_proxycode)
public static void phaseTwo()
{
    try{
        // check the caches
        if((_lastSentBytes == -1) && (_lastReceivedBytes == -1)){
            loadBillingInfo();
        }

        // get total
        _lastSentBytes    += _totalSentBytes;
        _lastReceivedBytes += _totalReceivedBytes;

        // send billing info
        autoSendBillingInfo();

        // successfull sending data so clear the rms
        clearRecordStore();
    }
    catch(Exception e){}
}

/**
 * This method will load the packet base billing record to the cache and
 * keep the billing info in the record store
 * This method will be used by the phaseOne and phaseTwo
 */

private static void loadBillingInfo()
{
    try {
        synchronized (_synchronizedObj) {
            RecordStore recordStore =
                RecordStore.openRecordStore(RECORD_STORE_NAME, true);

            int id = recordStore.getNextRecordID() - 1;
            byte [] record = recordStore.getRecord(id);
            ByteArrayInputStream bis = new ByteArrayInputStream(record);
            DataInputStream dis = new DataInputStream(bis);

            // load the billing info
            _lastReceivedBytes = dis.readLong();
            _lastSentBytes = dis.readLong();
            _recordStoreTime = dis.readLong();

            // close input stream and record store
            // don't need to close ByteArrayInputStream
            recordStore.closeRecordStore();
        }
    }

    catch(Exception e) {
        // there is not thing in the record store. Give an initialization data
        _lastReceivedBytes = 0;
        _lastSentBytes = 0;
        _recordStoreTime = System.currentTimeMillis();
    }
}

/**
 * Save new billing info rms + heap then delete the old record and update
 * the caches

```

Appendix B (send_proxycode)

```

*/
private static void saveBillingInfo()
    throws RecordStoreNotFoundException, RecordStoreException,
    IOException, RecordStoreFullException
{
    RecordStore recordStore = null;
    byte [] record = null;
    synchronized(_synchronizedObj) {
        // save the new info
        recordStore = RecordStore.openRecordStore(RECORD_STORE_NAME, true);

        ByteArrayOutputStream bos = new ByteArrayOutputStream(24);
        DataOutputStream dos = new DataOutputStream(bos);

        // save new received bytes = last received + heap
        dos.writeLong(_lastReceivedBytes + _totalReceivedBytes);

        // save new sent bytes = last sent + heap
        dos.writeLong(_lastSentBytes + _totalSentBytes);

        // save time
        dos.writeLong(_recordStoreTime);

        // save
        record = bos.toByteArray();
        recordStore.addRecord(record, 0, record.length);

        // already saved new record so update last info
        _lastReceivedBytes += _totalReceivedBytes;
        _lastSentBytes += _totalSentBytes;

        // now clear the heap
        clearHeap();

        // already saved new record so delete the old one
        if(recordStore.getNumRecords() > 2) {
            recordStore.deleteRecord(recordStore.getNextRecordID() - 2);
        }

        // close output stream and record store
        // don't need to close ByteArrayOutputStream
        recordStore.closeRecordStore();
    }
}

/**
 * Send the packet base billing info to MAS. After successful sending
 * billing record, clear all the record (heap and record store)
 * This method is used by the sendBillingInfo and saveBillingInfo
 */
private static boolean autoSendBillingInfo() throws IOException
{
    if(_lastReceivedBytes <=0 && _lastSentBytes <=0) {
        return true;
    }

    String es = "&";
    String eq = "=";
    if(ESCAPE_URL != 0) {

```

Appendix B (send_proxycode)

```

    es      = "%26";
    eq      = "%3D";
}

StringBuffer buff = new StringBuffer();
// append url
buff.append(MAS_PACKET_BASE_BILLING_URL);
buff.append(es);

// append total bytes sent
buff.append("sent" + eq);
buff.append(_lastSentBytes);
buff.append(es);

// append total bytes received
buff.append("received" + eq);
buff.append(_lastReceivedBytes);

String request    = buff.toString();
System.out.println(request);

// try to send billing info for 3 times.
int numOfRetry = 0;
while(numOfRetry < 3) {
    try {
        HttpURLConnection conn = (HttpURLConnection) Connector.open(request);
        InputStream is = conn.openInputStream();

        // close input and connection
        is.close();
        conn.close();
        return true;
    }
    catch(Exception e) {
        numOfRetry++;
    }
}

// we don't need to check for response if the http connection fail, it
// will through exception
return false;
}

private static void clearRecordStore()
{
    synchronized(_synchronizedObj) {
        try{
            _lastReceivedBytes    = 0;
            _lastSentBytes        = 0;
            RecordStore recordStore =
                RecordStore.openRecordStore(RECORD_STORE_NAME, false);
            recordStore.deleteRecord(recordStore.getNextRecordID() - 1);
            recordStore.closeRecordStore();

            // reset the record store time
            _recordStoreTime = System.currentTimeMillis();
        }
        catch(Exception e){}
    }
}

// The local variables used in the above section of code

```

Appendix B (send_proxycode)

```

public static long      _totalSentBytes      = 0;
public static long      _lastSentBytes       = -1;
public static long      _totalReceivedBytes  = 0;
public static long      _lastReceivedBytes   = -1;
public static Object    _synchronizedObj    = new Object();
public static long      _time                = 0;
public static long      _recordStoreTime     = 0;
public static Vector    _networkConnection  = new Vector(3);

```

```

////////////////////////////////////
//////// Public Constructor

```

```

public BillingOutputStream(OutputStream out)
{
    _outputStream = out;
}

```

```

////////////////////////////////////
//////// Public Members (Access Methods)

```

```

public void write(int b) throws IOException
{
    _outputStream.write(b);
    _totalSentBytes++;
    increment();
}

```

```

public void write(byte [] b) throws IOException
{
    _outputStream.write(b);
    _totalSentBytes += b.length;
    increment();
}

```

```

public void write(byte [] b, int off, int len) throws IOException
{
    _outputStream.write(b, off, len);
    _totalSentBytes += len;
    increment();
}

```

```

public void flush() throws IOException
{
    _outputStream.flush();
}

```

```

public void close() throws IOException
{
    // save the total sent bytes
    if(_totalSentBytes > 0) {
        PacketBaseBilling.incrementSentBytes(_totalSentBytes);
    }
    // reset the total sent bytes in case close() is called again
    _totalSentBytes = 0;
    _outputStream.close();
}

```

```

private void increment()
{
    if(_totalSentBytes > PACKET) {
        PacketBaseBilling.incrementSentBytes(_totalSentBytes);
    }
}

```

```

        _totalSentBytes = 0;
    }
}

```

Appendix B (send_proxycode)

```

////////////////////////////////////
//////// Private Fields
private OutputStream    _outputStream;
private long            _totalSentBytes = 0;
private final static int PACKET        = 10;

```

2005-09-1 02:26:02

Appendix C (receive_proxycode).txt

=====

Note: The pseudo-code in this file are exerpts from the original code file that manages receiving and sending of the data.

=====

```

/*
 * This proxy code allows original connection to get data and then
 * increments the internal counter to record it.
 * conn -- Original connection created for communicating with host
 * dgram -- The datagram that's to be sent.
 */

public static void receive(DatagramConnection conn, Datagram dgram)
throws IOException
{
    conn.receive(dgram);
    incrementReceivedBytes(dgram.getLength());
}

public static void incrementReceivedBytes(long bytes)
{
    synchronized(_synchronizedObj) {
        _totalReceivedBytes += bytes;

        // we always save billing info at the first time
        if((System.currentTimeMillis() - _time) > 60*1000) {
            // save billing info
            phaseOne();
            _time = System.currentTimeMillis();
        }
    }
}

/**
 * This phase will save the heap info to rms
 */

private static void phaseOne()
{
    try {
        // check the caches
        if((_lastSentBytes == -1) && (_lastReceivedBytes == -1)){
            loadBillingInfo();
        }

        // save new billing info and update caches
        saveBillingInfo();

        // do autosend this code can be taken out depend on device
        if(System.currentTimeMillis() - _recordStoreTime > 24*60*60*1000) {
            phaseTwo();
        }
    }
    catch(Exception e){}
}

/**
 * Send billing info from rms to MAS server
 */
public static void phaseTwo()
{

```

Appendix C (receive_proxycode).txt

```

try{
    // check the caches
    if((_lastSentBytes == -1) && (_lastReceivedBytes == -1)){
        loadBillingInfo();
    }

    // get total
    _lastSentBytes += _totalSentBytes;
    _lastReceivedBytes += _totalReceivedBytes;

    // send billing info
    autoSendBillingInfo();

    // successfull sending data so clear the rms
    clearRecordStore();
}
catch(Exception e){}
}

/**
 * This method will load the packet base billing record to the cache and
 * keep the billing info in the record store
 * This method will be used by the phaseOne and phaseTwo
 */

private static void loadBillingInfo()
{
    try {
        synchronized (_synchronizedObj) {
            RecordStore recordStore =
                RecordStore.openRecordStore(RECORD_STORE_NAME, true);

            int id = recordStore.getNextRecordID() - 1;
            byte [] record = recordStore.getRecord(id);
            ByteArrayInputStream bis = new ByteArrayInputStream(record);
            DataInputStream dis = new DataInputStream(bis);

            // load the billing info
            _lastReceivedBytes = dis.readLong();
            _lastSentBytes = dis.readLong();
            _recordStoreTime = dis.readLong();

            // close input stream and record store
            // don't need to close ByteArrayInputStream
            recordStore.closeRecordStore();
        }
    }

    catch(Exception e) {

        // there is not thing in the record store. Give an initialization data
        _lastReceivedBytes = 0;
        _lastSentBytes = 0;
        _recordStoreTime = System.currentTimeMillis();
    }
}

/**
 * Save new billing info rms + heap then delete the old record and update
 * the caches
 */

```

```

Appendix C (receive_proxycode).txt
private static void saveBillingInfo()
    throws RecordStoreNotFoundException, RecordStoreException,
        IOException, RecordStoreFullException
{
    RecordStore recordStore = null;
    byte [] record = null;
    synchronized(_synchronizedObj) {
        // save the new info
        recordStore = RecordStore.openRecordStore(RECORD_STORE_NAME, true);

        ByteArrayOutputStream bos = new ByteArrayOutputStream(24);
        DataOutputStream dos = new DataOutputStream(bos);

        // save new received bytes = last received + heap
        dos.writeLong(_lastReceivedBytes + _totalReceivedBytes);

        // save new sent bytes = last sent + heap
        dos.writeLong(_lastSentBytes + _totalSentBytes);

        // save time
        dos.writeLong(_recordStoreTime);

        // save
        record = bos.toByteArray();
        recordStore.addRecord(record, 0, record.length);

        // already saved new record so update last info
        _lastReceivedBytes += _totalReceivedBytes;
        _lastSentBytes     += _totalSentBytes;

        // now clear the heap
        clearHeap();

        // already saved new record so delete the old one
        if(recordStore.getNumRecords() > 2) {
            recordStore.deleteRecord(recordStore.getNextRecordID() - 2);
        }

        // close output stream and record store
        // don't need to close ByteArrayOutputStream
        recordStore.closeRecordStore();
    }
}

```

```

/**
 * Send the packet base billing info to MAS. After successful sending
 * billing record, clear all the record (heap and record store)
 * This method is used by the sendBillingInfo and saveBillingInfo
 */

```

```

private static boolean autoSendBillingInfo() throws IOException
{
    if(_lastReceivedBytes <=0 && _lastSentBytes <=0) {
        return true;
    }

    String es = "&";
    String eq = "=";
    if(ESCAPE_URL != 0) {
        es = "%26";
        eq = "%3D";
    }
}

```

Appendix C (receive_proxycode).txt

```

}

StringBuffer buff = new StringBuffer();
// append url
buff.append(MAS_PACKET_BASE_BILLING_URL);
buff.append(es);

// append total bytes sent
buff.append("sent" + eq);
buff.append(_lastSentBytes);
buff.append(es);

// append total bytes received
buff.append("received" + eq);
buff.append(_lastReceivedBytes);

String request      = buff.toString();
System.out.println(request);

// try to send billing info for 3 times.
int numOfRetry = 0;
while(numOfRetry < 3) {
    try {
        HttpURLConnection conn = (HttpURLConnection) Connector.open(request);
        InputStream is = conn.openInputStream();

        // close input and connection
        is.close();
        conn.close();
        return true;
    }
    catch(Exception e) {
        numOfRetry++;
    }
}

// we don't need to check for response if the http connection fail, it
// will through exception
return false;
}

private static void clearRecordStore()
{
    synchronized(_synchronizedObj) {
        try{
            _lastReceivedBytes      = 0;
            _lastSentBytes          = 0;
            RecordStore recordStore =
                RecordStore.openRecordStore(RECORD_STORE_NAME, false);
            recordStore.deleteRecord(recordStore.getNextRecordID() - 1);
            recordStore.closeRecordStore();

            // reset the record store time
            _recordStoreTime = System.currentTimeMillis();
        }
        catch(Exception e){}
    }
}

// The local variables used in the above section of code

```

```

Appendix C (receive_proxycode).txt
public static long      _totalSentBytes      = 0;
public static long      _lastSentBytes       = -1;
public static long      _totalReceivedBytes  = 0;
public static long      _lastReceivedBytes   = -1;
public static Object     _synchronizedObj    = new Object();
public static long       _time                = 0;
public static long       _recordStoreTime    = 0;
public static Vector     _networkConnection  = new Vector(3);

```

```

public class BillingInputStream extends InputStream
{

```

```

    public BillingInputStream(InputStream is)
    {
        _is = is;
    }

```

```

    //////////////////////////////////////
    ////////////////////////////////////// Public Members (Access Methods)
    //////////////////////////////////////

```

```

    public int read() throws IOException
    {
        int r = _is.read();
        if(r != -1) {
            _totalReceivedBytes++;
        }
        increment();
        return r;
    }

```

```

    public int available() throws IOException
    {
        return _is.available();
    }

```

```

    public void mark(int readlimit)
    {
        _is.mark(readlimit);
    }

```

```

    public boolean markSupported()
    {
        return _is.markSupported();
    }

```

```

    public int read(byte [] b) throws IOException
    {
        int i = _is.read(b);
        _totalReceivedBytes += i;
        increment();
        return i;
    }

```

```

    public int read(byte [] b, int off, int len) throws IOException
    {
        int i = _is.read(b, off, len);
        _totalReceivedBytes += i;
        increment();
        return i;
    }

```

```

    public long skip(long n) throws IOException
    {

```

```

Appendix C (receive_proxycode).txt
    return _is.skip(n);
}

public void reset() throws IOException
{
    _is.reset();
}

public void close() throws IOException
{
    // save the total received bytes
    if(_totalReceivedBytes > 0) {
        PacketBaseBilling.incrementReceivedBytes(_totalReceivedBytes);
    }

    // reset the total received bytes in case lose() is called again
    _totalReceivedBytes = 0;
    _is.close();
}

private void increment()
{
    if(_totalReceivedBytes > PACKET) {
        PacketBaseBilling.incrementReceivedBytes(_totalReceivedBytes);
        _totalReceivedBytes = 0;
    }
}

////////////////////////////////////
//////// Private Fields

private InputStream      _is;
private long             _totalReceivedBytes = 0;
private final static int PACKET           = 10;
}

```